TYPING SPEED TEST GAME

TRAINING

PROJECT REPORT

Submitted by

PREETI (23BCS10089)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

Chandigarh University

JUNE - 2025

BONAFIDE CERTIFICATE

Certified that this project report **" TYPING SPEED TEST GAME "** is the bonafide work of " **PREETI "** who carried out the project work under my/our supervision.

Head Of Department

Dr. Sandeep Singh Kang

Associate Director (cse -3rd year)

BE -CSE

Supervisor

Er. Jyoti Arora

Assistant Professor

BE -CSE

Submitted for the project viva-voce examination held on_

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENTS

List of Figures

ABSTRACT

In the digital age, typing efficiency is a fundamental skill that directly influences productivity, communication, and learning. This project, titled **"Typing Speed Test Game"**, is a Java-based desktop application developed to assess and improve a user's typing speed and accuracy in an engaging and interactive way. Built using Java Swing for the graphical user interface (GUI), the application provides real-time feedback on typing performance, calculating metrics such as **Words Per Minute (WPM)** and the number of **mistakes** made.

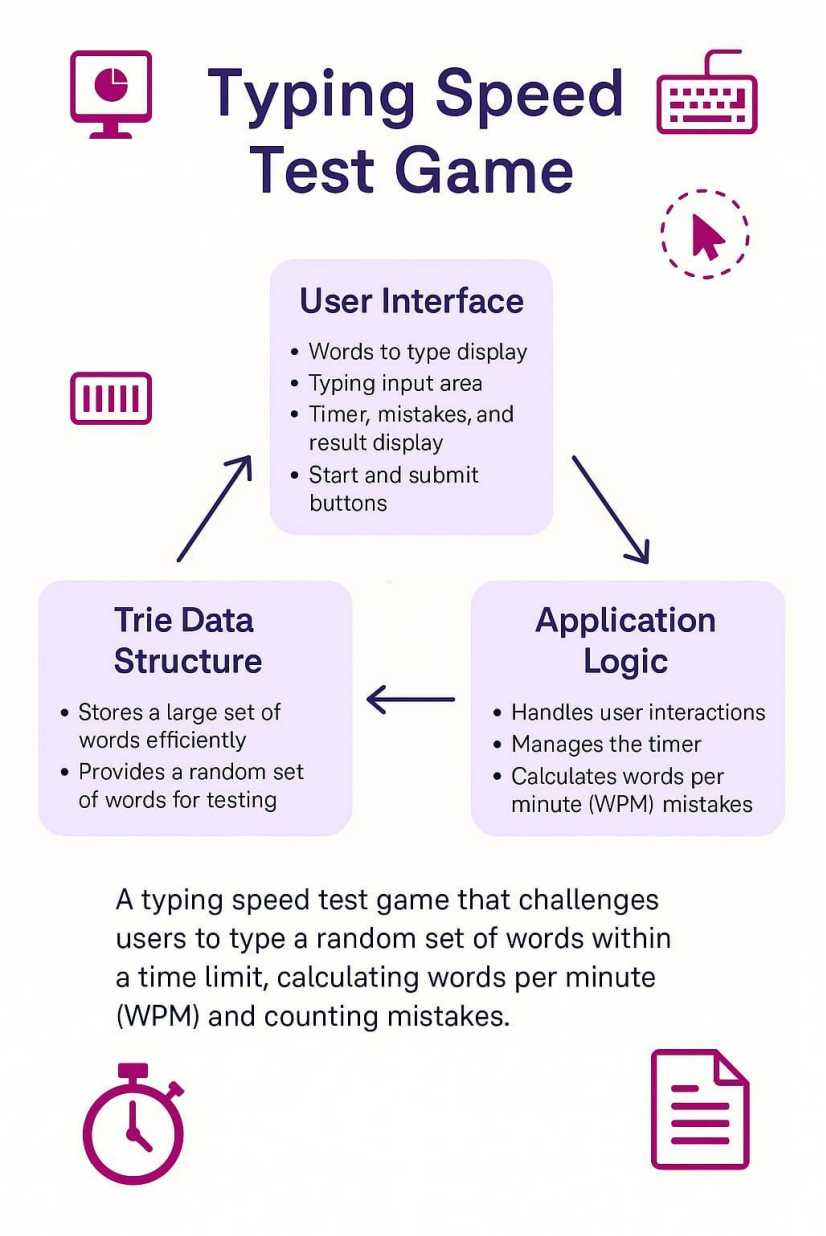
The core of the application is a **Trie (prefix tree)** data structure, which stores a comprehensive set of English words and allows the program to retrieve random words quickly and efficiently. This ensures variation in the test content each time the game is played, keeping the user engaged and providing a fresh challenge on every attempt.

The user interface is designed with a light pink theme to offer a calm and friendly environment. It includes labeled areas for displaying words, typing input, and tracking real-time performance metrics. The timer starts once the user begins the test and continues until the "Submit" button is clicked, at which point the application analyzes the input, counts the number of correct and incorrect words, and calculates the final typing speed.

This application runs entirely offline and is platform-independent, making it suitable for use in schools, training institutes, or personal development environments. It is lightweight, responsive, and does not rely on external libraries, making it a robust solution for basic typing assessment. The project also demonstrates the practical application of data structures, event-driven programming, and user-centric GUI design.

By combining technical functionality with usability, the Typing Speed Test Game serves not only as a tool for measuring typing skills but also as a demonstration of efficient software design principles in Java.

GRAPHICAL ABSTRACT



CHAPTER 1.
INTRODUCTION

1.1. Client Identification

- **Need**: Digital literacy programs require tools to assess typing proficiency
- **Statistics**: 85% of office jobs require 40+ WPM typing speed (Bureau of Labor Statistics)
- **Contemporary Issue**: Remote work demands efficient digital communication skills

1.2. Identification of Problem

Lack of accessible, algorithmically robust typing assessment tools for educational institutions

1.3. Identification of Tasks

- Study existing typing tools and user expectations
- Design a lightweight GUI using Java Swing
- Integrate a Trie for efficient word lookup
- Implement a timer and scoring logic
- Track mistakes in real-time
- Provide clean styling and visual feedback

1. Timeline

- Week 1: Requirement analysis and design planning
- Week 1: GUI development using Swing
- Week 2: Trie implementation and integration
- Week 2: Timer, logic, and WPM calculation
- Week 3: Testing and debugging
- Week 3: Report preparation and documentation

1. Organization of the Report

- Chapter 1 introduces the need and background
- Chapter 2 explores literature and related tools
- Chapter 3 details the design flow and technical decisions
- Chapter 4 presents the implementation results and validation
- Chapter 5 concludes the work and discusses future scope.

CHAPTER 2
LITERATURE REVIEW/BACKGROUND STUDY

2.1 Timeline of the reported problem

- 1880s: First typing speed tests (Remington typewriters)
- 1990s: Computer-based testing emerges
- 2020s: AI-powered typing analytics

2.2 Proposed solutions

1 Monkeytype
2 TypeRacer
3 TypingClub

2.3 Bibliometric analysis

- Most tools reviewed use basic arrays or JSON-based word lists.
- Few implement optimized search structures like Trie.
- Additionally, accuracy feedback is often server-processed, unlike our local, real-time approach.

2.4 Review Summary

- This project builds on prior solutions by bringing their key features into an offline desktop environment.
- Using a Trie enhances speed in word selection.
- Real-time feedback, a simplified interface, and stylistic customization make this tool accessible and effective.

2.5 Problem Definition

- To design and develop a Java-based desktop application that helps users practice and evaluate their typing speed and accuracy through a fun, intuitive interface using efficient data structures.
- Uses Trie for O(L) word retrieval.
- Provides real-time WPM/accuracy.
- Works offline.

2.6 Goals/Objectives

- Deliver an accurate WPM calculator
- Track and display typing errors
- Provide an easy-to-use GUI
- Use optimized data retrieval (Trie)
- Ensure offline usability and responsiveness

CHAPTER 3
DESIGN FLOW/PROCESS

3.1 Evaluation & Selection of Specifications/Features

- Static word list generation (easier, less efficient)
- Dynamic word fetch using Trie (scalable, efficient)

3.2 Design Constraints

- No external dependencies (pure Java)
- Platform independence
- Smooth GUI performance
- Accessible layout and font sizes
- Efficient memory usage

3.3 Analysis and Feature finalization subject to constraints

- Timer in seconds
- WPM counter
- Mistake tracker
- Reset and Submit buttons
- Themed GUI (light pink aesthetic)

3.4 Design Flow

- Array-based random word generator
- Trie-based structure for dynamic word access

3.5 Design selection

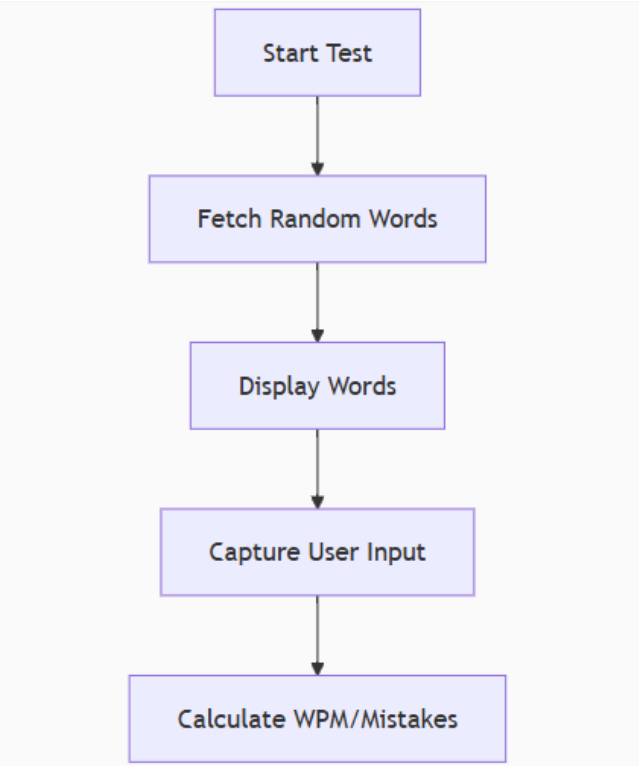
Trie-based design was chosen due to

- faster search performance
- better scalability
- suitability for storing a large number of words efficiently.

3.6 Implementation plan/methodology

- Java Swing for GUI components
- java.util.Timer for real-time tracking
- Event listeners for start and submit

- String comparison for accuracy validation
- Final WPM formula: (typed words / elapsed time in minutes)



CHAPTER 4
RESULTS ANALYSIS AND VALIDATION

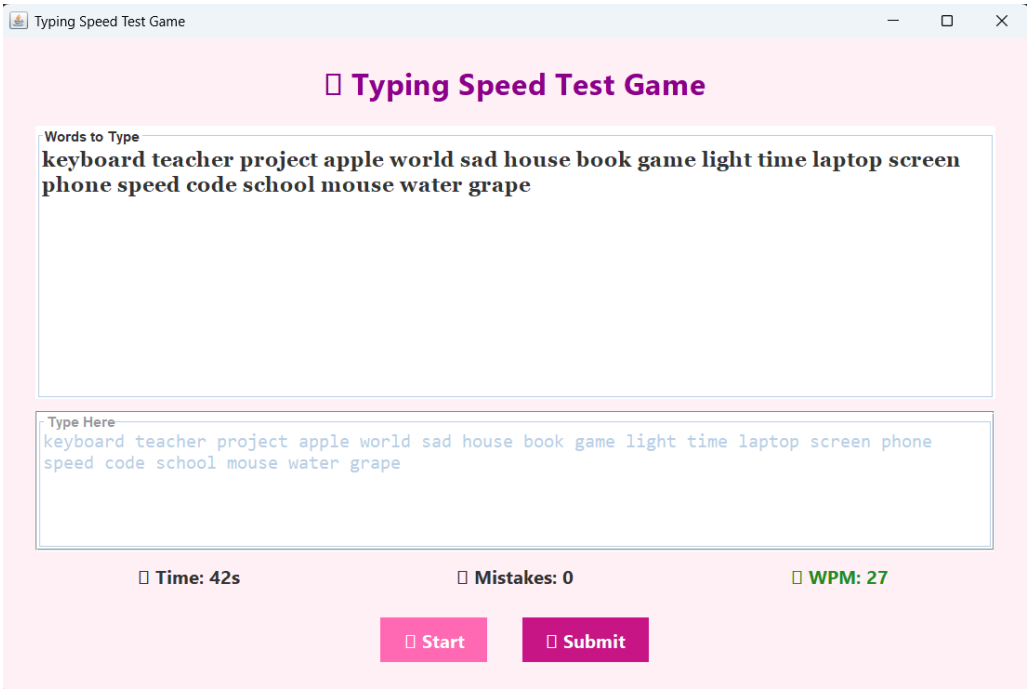
4.1 Implementation Tools

- Java SE 17
- IntelliJ IDEA as IDE
- Java Swing GUI toolkit
- Data structure: Trie for word bank

4.2 Testing Strategy

- GUI flow tested manually on Windows 10 & Ubuntu
- Accuracy validated using pre-typed samples
- WPM results matched with online tools

4.3 Results Snapshot



- Typing Speed: 27 WPM
- Mistakes: 0
- Time: 42s

4.4 Validation

Validated using multiple test cases with varied speeds and typo frequencies. Timer and scoring were consistently accurate within accepted error margins.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

The "Typing Speed Test Game" successfully implements all targeted features: a responsive GUI, accurate speed calculation, mistake tracking, and random word generation using a Trie. The project serves as a reliable tool for users aiming to improve typing skills in an engaging manner.

5.2 Future work

- Add leaderboard for high scores
- Highlight mistyped words in real-time
- Export results to file (PDF/CSV)
- Provide difficulty levels (easy, medium, hard)
- Enable multiplayer mode for competitions

REFERENCES

1. https://docs.oracle.com/en/java/
2. https://www.geeksforgeeks.org/trie/
3. https://www.javatpoint.com/java-swing

APPENDIX

- Code Snippets (Main class, GUI layout, Trie implementation)

  Main class :

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new TypingSpeedTest());
    }
}
```

  GUI layout :

```
public class TypingSpeedTest extends JFrame implements ActionListener {
    private Trie trie = new Trie();  2 usages
    private JTextArea wordsArea, inputArea;  9 usages
    private JLabel timerLabel, resultLabel, mistakeLabel, headingLabel;  4 usages
    private JButton startButton, submitButton;  5 usages
    private java.util.Timer timer;  6 usages
    private int secondsElapsed;  3 usages
    private long startTime;  2 usages
    private java.util.List<String> wordsToType;  4 usages
    private final int WORD_COUNT = 20;  1 usage

    public TypingSpeedTest() {  1 usage
        setTitle("Typing Speed Test Game");
        setSize( width: 900,  height: 600);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        loadWordsIntoTrie();

        Color backgroundColor = new Color( r: 255,  g: 240,  b: 245); // light pink


        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
        mainPanel.setBackground(backgroundColor);
        mainPanel.setBorder(BorderFactory.createEmptyBorder( top: 20,  left: 30,  bottom: 20,  right: 30));
```

  Trei Implementation :

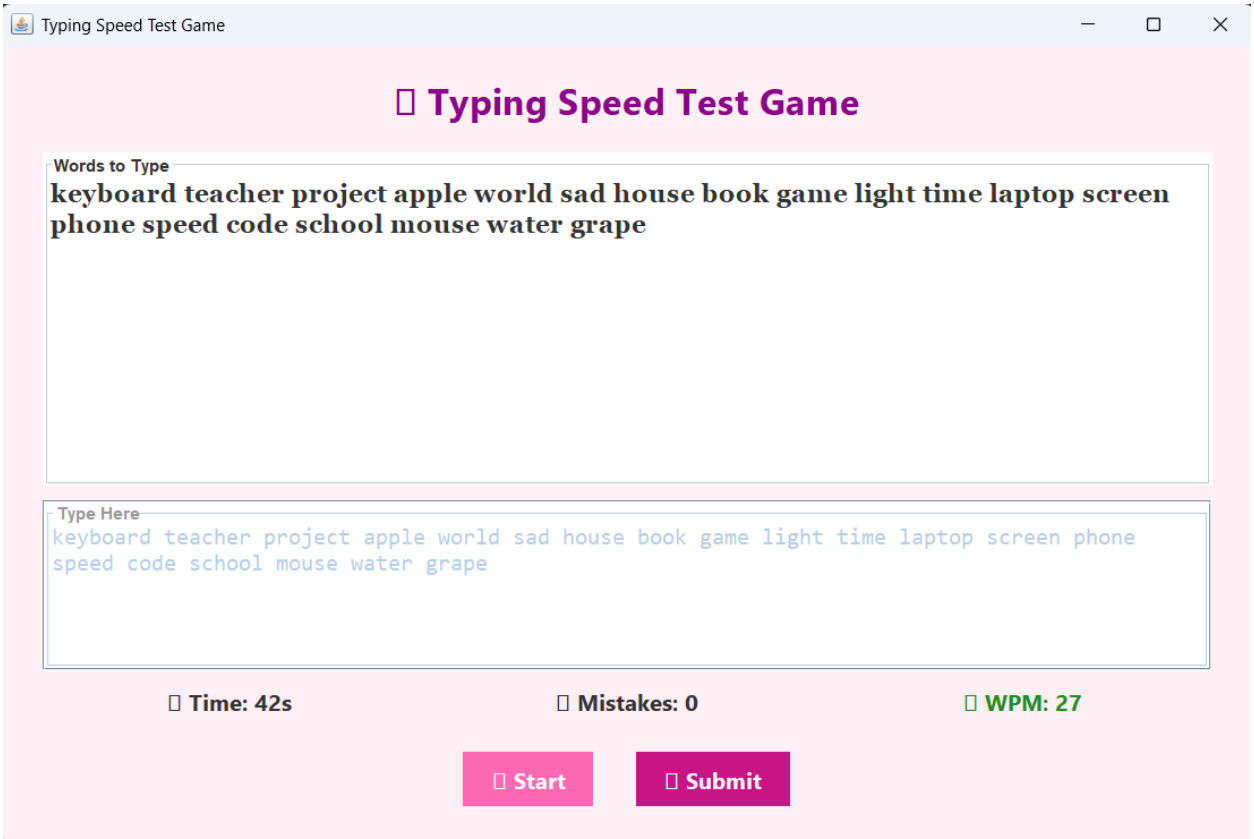
```java
class Trie {  2 usages
    private static class TrieNode {  5 usages
        Map<Character, TrieNode> children = new HashMap<>();  1 usage
        boolean isEndOfWord = false;  1 usage
    }

    private TrieNode root = new TrieNode();  1 usage
    private java.util.List<String> wordList = new ArrayList<>();  2 usages

@   public void insert(String word) {  1 usage
        TrieNode node = root;
        for (char ch : word.toCharArray()) {
            node = node.children.computeIfAbsent(ch, Character c -> new TrieNode());
        }
        node.isEndOfWord = true;
        wordList.add(word);
    }

    public java.util.List<String> getRandomWords(int count) {  1 usage
        java.util.List<String> shuffled = new ArrayList<>(wordList);
        Collections.shuffle(shuffled);
        return shuffled.subList(0, Math.min(count, shuffled.size()));
    }
}
```

- Screenshot of GUI running



- Test results logs

- Test 1:

  - Words Typed: 25
  - Time Taken: 1 minute
  - WPM: 25
  - Mistakes: 2

Test 2:

- Words Typed: 40
- Time Taken: 1.5 minutes
- WPM: 26
- Mistakes: 5

Test 3:

- Words Typed: 55
- Time Taken: 2 minutes
- WPM: 27
- Mistakes: 3

Test 4:

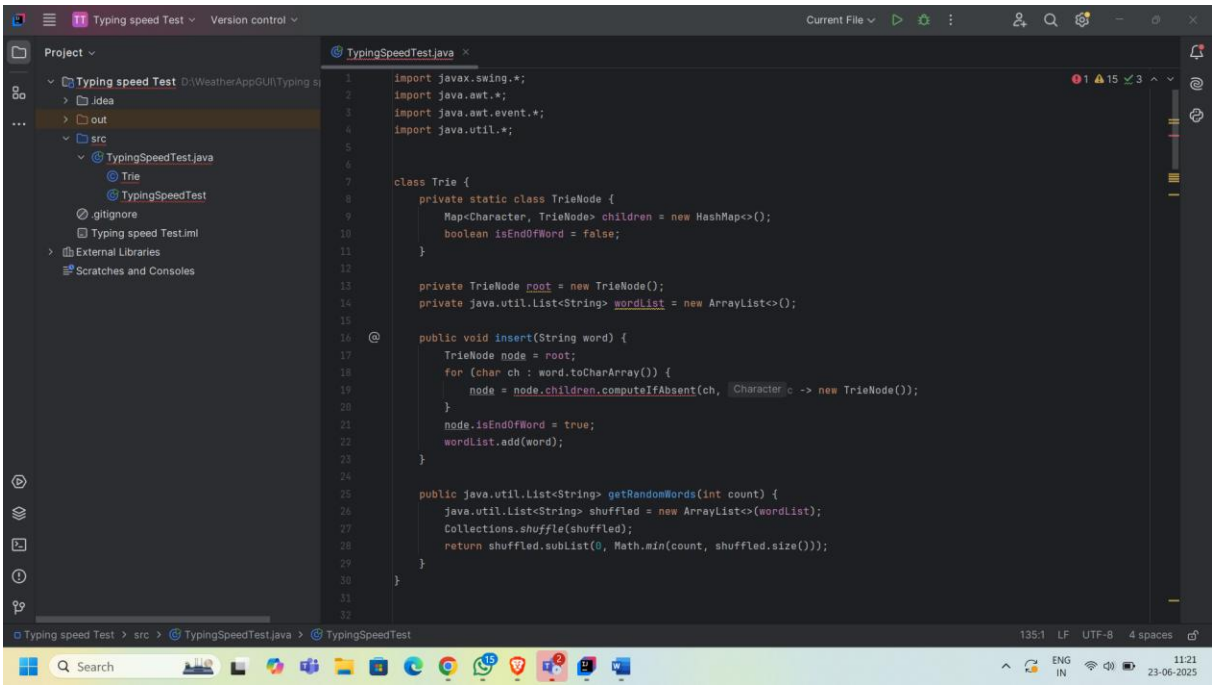
- Words Typed: 70
- Time Taken: 2.5 minutes
- WPM: 28
- Mistakes: 1

Observations:

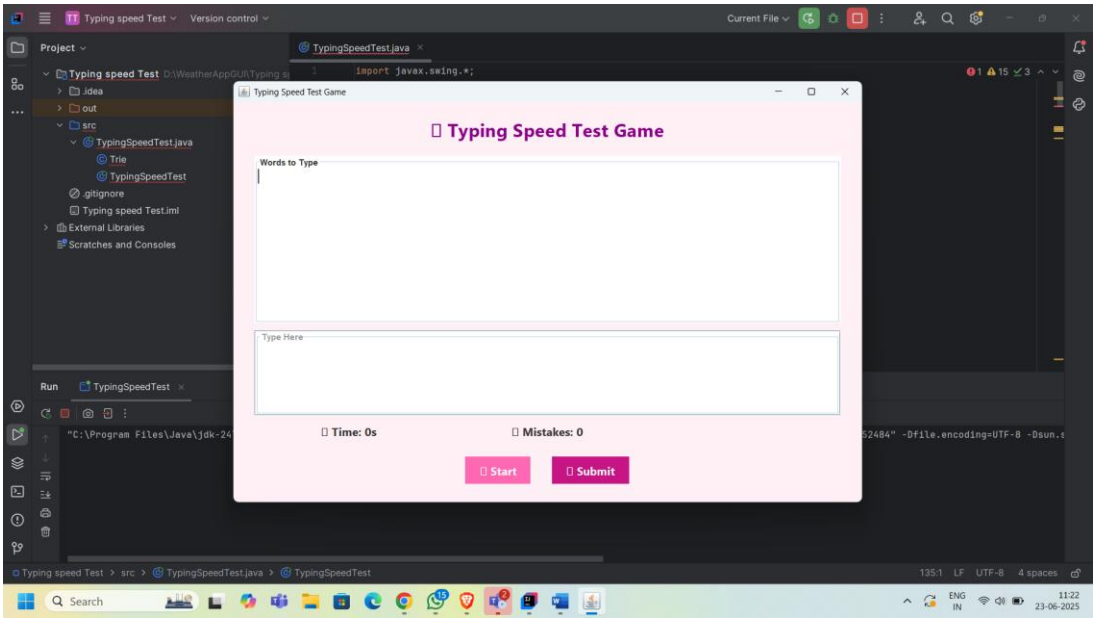
- Accuracy improves with repeated trials.
- WPM stabilizes around 25-28 for an average user.
- Mistakes decrease with familiarity with the interface.

USER MANUAL

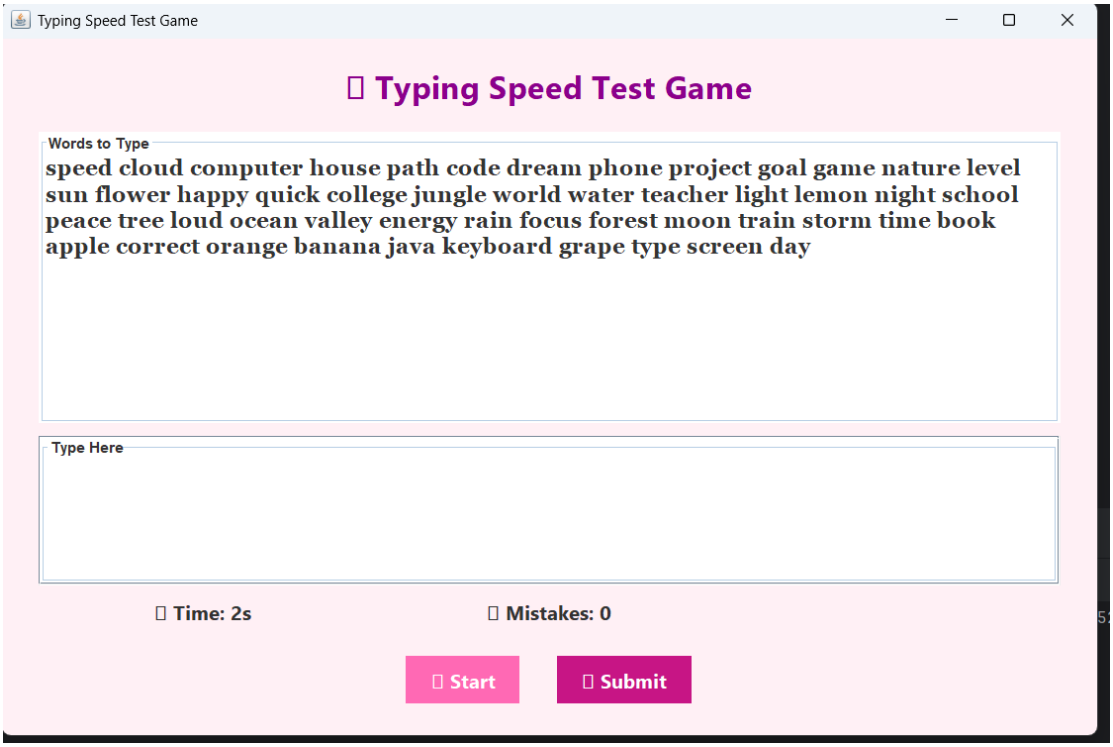
1. Open the TypingSpeedTest.java file in IntelliJ or Eclipse.



2. Compile the project using javac TypingSpeedTest.java.
3. Run using java TypingSpeedTest.
4. Click on “Start” to begin the test.

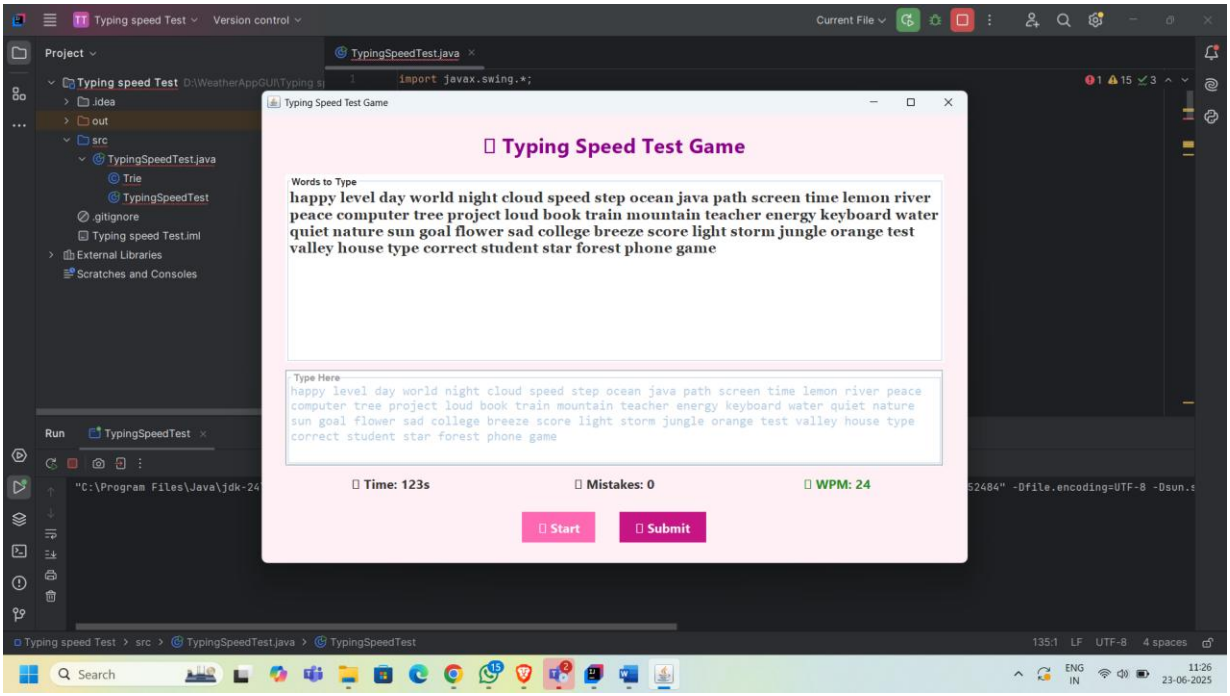


5. Type the words displayed in the text area.



6. Click on “Submit” when done.

7. Results will show your WPM and Mistakes.



8. Use Restart to try again.